MM_Event

This module is part of the original MMBasic library. It is reproduced here with kind permission of Hugh Buckle and Geoff Graham. Be aware it may reference functionality which has changed or is deprecated in the latest versions of MMBasic.

SE02.BAS:

```
'File....: SE02.Bas
'
'Author..: Simon Whittam
'
'Email...: s.whittam(at)xnet.co.nz
'
'Date....: 8th September 2013
'
'Language: Maximite BASIC, v4.4
'
'Purpose.: To provide an example of a Event Driven framework that:
'            uses circular First In, First Out (FIFO) event queues comprised
of String Arrays.
'            debounces switch inputs.
'            uses Bit flags stored in a numeric variable.
'            is comprised of two co-operating Finite State Machines (FSM).
'            uses non blocking timers (10ms and 1sec) for delays.
'            uses events to move between states and communicate between
FSM's.
'            displays State/Event changes
'            the use of pointer to a SUB (ON var GOSUB A0,A1,A2,...)
'
'          The code example turns on & off the toggling of a LED only when a
pushbutton
'          switch has been pushed 3 times and remains pushed, all within 2
seconds.
'
'Version.: v1.1
'
'License.: Attribution-NonCommercial-ShareAlike 3.0 Australia (CC BY-NC-SA
3.0 AU)
'
'Ref.....: http://www.state-machine.com/qm/
'          http://geoffg.net/MonoMaximite.html
'          http://mmbasic.com/
'          http://creativecommons.org/licenses/by-nc-sa/3.0/au/
'
'=============================================================================
===

Library Load "Bit.Lib"
```

```
'BEGIN EventQ1.Lib - Initialisation

   Library Load "EventQ1.Lib"

   Option BASE 0     'event queue data referenced as an offset from: 0 -
(Length -1)

   'Configure event queue as empty
   EQ1Length    = 4  'Event queue length.
   EQ1WriteFlag = 1  'Gets toggled after writing to last location in event
queue.
   EQ1ReadFlag  = 1  'Gets toggled after reading last location in event
queue.
   EQ1NextWrite = 0  'Next location to write a byte.
   EQ1NextRead  = 0  'Next location to read a byte.

   Dim EQ1$(1) LENGTH EQ1Length  'Create a circular event queue using a
string/byte array.

   EQ1$ = "FIFO"

'END EventQ1.Lib - Initialisation

'=============================================================================
===

'BEGIN EventQ2.Lib - Initialisation

   Library Load "EventQ2.Lib"

   'Option BASE 0     'event queue data referenced as an offset from: 0 -
(Length -1)

   'Configure event queue as empty
   EQ2Length    = 4  'Event queue length.
   EQ2WriteFlag = 1  'Gets toggled after writing to last location in event
queue.
   EQ2ReadFlag  = 1  'Gets toggled after reading last location in event
queue.
   EQ2NextWrite = 0  'Next location to write a byte.
   EQ2NextRead  = 0  'Next location to read a byte.

   Dim EQ2$(1) LENGTH EQ2Length  'Create a circular event queue using a
string/byte array.

   EQ2$ = "FIFO"

'END EventQ2.Lib - Initialisation

'=============================================================================
===
```

```
'Define I/O pins used
Pin(11) = 1      ' Pin 11 -> 1, LED off

SetPin 12, 2     ' Pin 12, Input, 5v Digital, with external pull-up resistor
SetPin 11, 9     ' Pin 11, Output, Open Collector, with LED & resistor to
5Vdc.


'SysFlags bit assignments
'  SFTick10ms = 0
'  SFTick1s   = 1
'  SFSwitch   = 2
'  SFLED      = 3

SysFlags     = &B0000


' Queue1 event definitions used
'   Undefined1     = 0
'   Switch1.0      = 1
'   Switch1.1      = 2
'   TickTimeout1   = 3
'   SecTimeout1    = 4

' Queue2 event definitions used
'   Undefined2   = 0
'   TickTimeout2 = 1
'   Enable2      = 2
'   Disable2     = 3

'
SetTick   10, BitSet10ms, 1
SetTick 1000, BitSet1s  , 2

' Queue1 variables
'=================
SecTimeout1          = 0
TickTimeout1         = 0
Switch1DebounceCount = 0
CurState1            = 0  ' Current State
PrvState1            = 0  ' Previous State
CurEvent1            = 0  ' Undefined event
CurStateEvent1       = 0
PrvStateEvent1       = 0

' Queue2 variables
'=================
TickTimeout2         = 0
CurState2            = 0  ' Current State
PrvState2            = 0  ' Previous State
CurEvent2            = 0  ' Undefined event
```

```
CurStateEvent2       = 0
PrvStateEvent2       = 0

Do While (1)

'=============================================================================

   'BEGIN Debug1
   '  PrvStateEvent1 = CurStateEvent1
   'END Debug1


   'Respond to events from Queue1 for Finite State Machine 1
   CurEvent1      = EQ1Read()
   CurStateEvent1 = ( Fix( (CurState1 * 5) + CurEvent1 ) + 1 )


   'BEGIN Debug1
   '  Print only first occurrence of State/Event change
   '  IF NOT( PrvStateEvent1 = CurStateEvent1 ) THEN
   '     PRINT "StateEvent1: " + STR$( CurStateEvent1 )
   '  ENDIF
   'END Debug1


   On CurStateEvent1 GoSub A0,A1,A2,A3,A4,B0,B1,B2,B3,B4

'=============================================================================

   'BEGIN Debug2
   '  PrvStateEvent2 = CurStateEvent2
   'END Debug2


   'Respond to events from Queue2 for Finite State Machine 2
   CurEvent2      = EQ2Read()
   CurStateEvent2 = ( Fix( (CurState2 * 4) + CurEvent2 ) + 1 )


   'BEGIN Debug2
   '  Print only first occurrence of State/Event change
   '  IF NOT( PrvStateEvent2 = CurStateEvent2 ) THEN
   '     PRINT "StateEvent2: " + STR$( CurStateEvent2 )
   '  ENDIF
   'END Debug2


   On CurStateEvent2 GoSub E0,E1,E2,E3,F0,F1,F2,F3,G0,G1,G2,G3

'=============================================================================
```

```
   'Has SFTick10ms flag been set
   IF BitTstSet( SysFlags, 0 )  THEN

      DecTimeOut1( TickTimeout1, 3)

      DecTimeout2( TickTimeout2, 1)

      'DecTimeoutN( TickTimeoutN, TickTimeoutEventN)

      DebounceSwitch1

      'DebounceSwitchN


      SysFlags = BitClr( SysFlags, 0 )  'Clear SFTick10ms flag.

   ENDIF


   'Has SFTick1s flag been set
   If BitTstSet( SysFlags, 1 )  Then

      DecTimeout1( SecTimeout1, 4 )

      'DecTimeoutN( SecTimeoutN, TimeoutNEvent )

      SysFlags = BitClr( SysFlags, 1 )  'Clear SFTick1s flag.

   ENDIF

Loop

Print "Finished"

End

'================================================================================
===

BitSet10ms:

   'GLOBAL: SysFlags

   'Set SFTick10ms flag in SysFlags
   SysFlags = BitSet( SysFlags, 0 )

IReturn

'================================================================================
===
```

```
BitSet1s:

   'GLOBAL: SysFlags

   'Set SFTick1s flag in SysFlags
   SysFlags = BitSet( SysFlags, 1 )

IReturn

'================================================================================
===

Sub DecTimeout1( TimeoutCount, TimeoutEvent)

   'Has Timeout already expired ?
   If TimeoutCount > 0  Then

      ' No, decrement Timeout count
      TimeoutCount = ( TimeoutCount - 1 )

      'Has Timeout delay expired ?
      If TimeoutCount = 0  Then

         ' Yes, add Timeout event to queue 1.
         EQ1WriteSuccess( TimeoutEvent )

      ENDIF

   ENDIF

End Sub   'DecTimeout1

;================================================================================
===

Sub DecTimeout2( TimeoutCount, TimeoutEvent)

   'Has Timeout already expired ?
   If TimeoutCount > 0  Then

      ' No, decrement Timeout count
      TimeoutCount = ( TimeoutCount - 1 )

      'Has Timeout delay expired ?
      If TimeoutCount = 0  Then

         ' Yes, add Timeout event to queue 2.
         EQ2WriteSuccess( TimeoutEvent )

      ENDIF
```

```
   ENDIF

End Sub   'DecTimeout2

;=============================================================================
===

Sub DebounceSwitch1

   'GLOBAL: Switch1DebounceCount, SysFlags

   ' Is a switch state being debounced ?
   If Switch1DebounceCount = 0  Then

      'No, Check for change of switch state.
      Local SwitchPin, SwitchBit

      SwitchBit = BitMask( SysFlags, 4 ) ' Mask SFSwitch flag, i.e. 2^2.

      SwitchPin = Pin(12)
      SwitchPin = BitToggle( SwitchPin, 0 )     ' Inputs are active low
      SwitchPin = BitShiftLeft( SwitchPin )     ' Match PIN bit position
with position of SFSwitch bit in SysFags
      SwitchPin = BitShiftLeft( SwitchPin )

      ' Has switch changed state ?
      If Not( SwitchPin = SwitchBit )  Then

         'Yes, debounce switch1.
         SwitchDebounced( Switch1DebounceCount, 2, 2, 1 )

      ENDIF

   Else       'Switch1DebounceCount <> 0
      SwitchDebounced( Switch1DebounceCount, 2, 2, 1 )

   ENDIF

End Sub   'DebounceSwitch1

;=============================================================================
===

Sub SwitchDebounced( SwitchDebounceCount, SwitchBit, Switch1Event,
Switch0Event )

   'GLOBAL: SysFlags

   SwitchDebounceCount = SwitchDebounceCount + 1

   'Has 50ms (5 x 10ms) of debounce time elapsed ?
```

```
   If ( SwitchDebounceCount = 5 )   Then

      SysFlags = BitToggle( SysFlags, SwitchBit)   ' Yes, toggle SFSwitch
bit in SysFlags

      If BitTstSet( SysFlags, SwitchBit )  Then
         EQ1WriteSuccess( Switch1event )           ' Place Switch1 event in
queue.
      Else
         EQ1WriteSuccess( Switch0Event )           ' Place Switch0 event in
queue.
      ENDIF

      SwitchDebounceCount = 0

   ENDIF

End Sub   'SwitchDebounced

;=============================================================================
===

A0:   'Q1, 1, State0 - Undefined

Return

;=============================================================================
===

A1:   'Q1, 2, State0 - Switch0

Return

;=============================================================================
===

A2:   'Q1, 3, State0 - Switch1

   PrvState1 = CurState1
   CurState1 = 1

   SecTimeout1  = 2
   Switch1Count = 1

Return

;=============================================================================
===

A3:   'Q1, 4, State0 - TickTimeout
```

```
Return

;================================================================================
===

A4:   'Q1, 5, State0 - SecondTimeout

Return

;================================================================================
===

B0:   'Q1, 6, State1 - Undefined

Return

;================================================================================
===

B1:   'Q1, 7, State1 - Switch0

Return

;================================================================================
===

B2:   'Q1, 8, State1 - Switch1

   Switch1Count = (Switch1Count + 1 )

Return

;================================================================================
===

B3:   'Q1, 9, State1 - TickTimeout

Return

;================================================================================
===

B4:   'Q1, 10, State1 - SecondTimeout

   'Has Switch1 been pressed twice and still pressed after 2 seconds
   If (Switch1Count = 3)  And  BitTstSet( SysFlags, 2 )  Then

      'Toggle LED state
      SysFlags = BitToggle( SysFlags, 3)   ' Toggle SFLed flag

      If BitTstSet( SysFlags, 3 )  THEN
```

```
      '  Pin(11) = 0           'Turn LED on
         EQ2WriteSuccess( 2 )  'Send Enable2 event to event queue 2
      Else
      '  Pin(11) = 1           'Turn LED off
         EQ2WriteSuccess( 3 )  'Send Disable2 event to event queue 2
      EndIf

   ENDIF


   Switch1Count = 0

   PrvState1    = CurState1
   CurState1    = 0

Return

;================================================================================
===

E0:   'Q2, 1, State0 - Undefined2

Return

;================================================================================
===

E1:   'Q2, 2, State0 - TickTimeout2

Return

;================================================================================
===

E2:   'Q2, 3, State0 - Enable2

   PrvState2 = CurState2
   CurState2 = 1

   TickTimeout2 = 20

   Pin(11) = 0    'Turn LED on

Return

;================================================================================
===

E3:   'Q2, 4, State0 - Disable2

Return
```

```
;================================================================================
===

F0:   'Q2, 5, State1 - Undefined2

Return

;================================================================================
===

F1:   'Q2, 6, State1 - TickTimeout2

   PrvState2 = CurState2
   CurState2 = 2

   TickTimeout2 = 20

   Pin(11) = 1    'Turn LED off

Return

;================================================================================
===

F2:   'Q2, 7, State1 - Enable2

Return

;================================================================================
===

F3:   'Q2, 8, State1 - Disable2

   PrvState2 = CurState2
   CurState2 = 0

   Pin(11) = 1    'Turn LED off

Return

;================================================================================
===

G0:   'Q2, 9, State2 - Undefined2

Return

;================================================================================
===

G1:   'Q2,10, State2 - TickTimeout2
```

```
   PrvState2 = CurState2
   CurState2 = 1

   TickTimeout2 = 20

   Pin(11) = 0    'Turn LED on

Return

;================================================================================
===

G2:   'Q2, 11, State2 - Enable2

Return

;================================================================================
===

G3:   'Q2, 12, State2 - Disable2

   PrvState2 = CurState2
   CurState2 = 0

   Pin(11) = 1    'Turn LED off

Return

;================================================================================
===
```

BIT.LIB:

```
'File....: Bit.Lib
'
'Author..: Simon Whittam
'
'Email...: s.whittam(at)xnet.co.nz
'
'Date....: 8th September 2013
'
'Language: Maximite BASIC, v4.4
'
'Purpose.: Provide bit manipulation functions of Bit Flags contained in a
'            MM BASIC numeric variable.
'
'Version.: v1.0
'
'License.: Attribution-NonCommercial-ShareAlike 3.0 Australia (CC BY-NC-SA
```

```
3.0 AU)
'
'Ref.....: MMBasic - User Library, BIN8.BAS, crackerjack
'          http://lbpe.wikispaces.com/Bit.Shift
'          https://en.wikipedia.org/wiki/Bit_manipulation
'          http://geoffg.net/MonoMaximite.html
'          http://mmbasic.com/
'          http://creativecommons.org/licenses/by-nc-sa/3.0/au/
'
'=============================================================================
===

FUNCTION BitComp( Nbr )
   BitComp = (-Nbr -1)
END FUNCTION

'=============================================================================
===

FUNCTION BitSet( Nbr, Bit )
   BitSet = ( Nbr OR 2^Bit )
END FUNCTION

'=============================================================================
===

FUNCTION BitClr( Nbr, Bit )
   BitClr = ( Nbr AND BitComp(2^Bit) )
END FUNCTION

'=============================================================================
===

FUNCTION BitTstSet( Nbr, Bit )
   BitTstSet = SGN( Nbr AND 2^Bit)
END FUNCTION

'=============================================================================
===

FUNCTION BitTstClr( Nbr, Bit )
   BitTstClr = SGN( BitComp(Nbr) AND 2^Bit)
END FUNCTION

'=============================================================================
===

FUNCTION BitShiftRight( Nbr)
    BitShiftRight = FIX( Nbr / 2)
END FUNCTION
```

```
'=============================================================================
===

FUNCTION BitMask( Nbr, Mask )
   BitMask = ( Nbr AND Mask )
END FUNCTION

'=============================================================================
===

FUNCTION BitToggle( Nbr, Bit)
   BitToggle = (Nbr XOR 2^Bit)
END FUNCTION

'=============================================================================
===

FUNCTION BitShiftLeft( Nbr)
    BitShiftLeft = ( Nbr * 2 )
END FUNCTION

'=============================================================================
===

FUNCTION BitRotLeft(Nbr)
    BitRotLeft = ((Nbr+Nbr) MOD 256) or (Nbr>127)
END FUNCTION

'=============================================================================
===

FUNCTION BitRotRight(Nbr)
    BitRotRight = (128*(Nbr AND 1)) or FIX(Nbr/2)
END FUNCTION

'=============================================================================
===
```

EVENTQ1.lib:

```
'File....: EventQ1.Lib
'
'Author..: Simon Whittam
'
'Email...: s.whittam(at)xnet.co.nz
'
'Date....: 8th September 2013
'
'Language: Maximite BASIC, v4.4
```

```
'
'Purpose.: Create and initialise an instance of a First In, First Out
circular event queue.
'          The event queue saves single bytes in a small String Array and
does not
'          overwrite unread bytes.
'
'          Intended for use with a event driven Finite State Machine.
'
'Version.: v1.1
'
'License.: Attribution-NonCommercial-ShareAlike 3.0 Australia (CC BY-NC-SA
3.0 AU)
'
'Ref.....: https://en.wikipedia.org/wiki/Circular_buffer
'          http://geoffg.net/MonoMaximite.html
'          http://mmbasic.com/
'          http://creativecommons.org/licenses/by-nc-sa/3.0/au/
'
'================================================================================
===

'BEGIN EventQ1.Lib - Initialisation

   'Copy the "EventQ1.Lib - Initialisation" section to the beginning of the
Main BASIC module.
   'Unremark the NEXT line when using this file as as library file
   'LIBRARY LOAD "EventQ1.Lib"

   OPTION BASE 0     'event queue data referenced as an offset from: 0 -
(Length -1)

   'Configure event queue as empty
   EQ1Length    = 4  'Event queue length.
   EQ1WriteFlag = 1  'Gets toggled after writing to last location in event
queue.
   EQ1ReadFlag  = 1  'Gets toggled after reading last location in event
queue.
   EQ1NextWrite = 0  'Next location to write a byte.
   EQ1NextRead  = 0  'Next location to read a byte.

   DIM EQ1$(1) LENGTH EQ1Length  'Create a circular event queue using a
string/byte array.

   EQ1$ = "FIFO"

'END EventQ1.Lib - Initialisation

'================================================================================
===
```

```
'BEGIN Test code

   'Note:
   '   For an empty queue, the following variables are initialised as
follows:
   '         EQ1WriteFlag = 1
   '         EQ1ReadFlag  = 1
   '         EQ1NextWrite = 0
   '         EQ1NextRead  = 0
   '


   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQWrite:" + STR$( EQ1Write( 49 ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQWrite:" + STR$( EQ1Write( 50 ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQWrite:" + STR$( EQ1Write( 51 ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQWrite:" + STR$( EQ1Write( 52 ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
'   PRINT "EQWrite:" + STR$( EQ1Write( 53 ) )
   EQ1WriteSuccess(53)
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQ1WriteFlag:" + STR$(EQ1WriteFlag) + ", EQ1ReadFlag:" +
STR$(EQ1ReadFlag)

   PRINT

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
```

```
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)
   PRINT "QueRead:" + STR$( EQ1Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)
   PRINT "QueRead:" + STR$( EQ1Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)
   PRINT "QueRead:" + STR$( EQ1Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)
   PRINT "QueRead:" + STR$( EQ1Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ1WriteFlag, EQ1ReadFlag,
EQ1NextWrite, EQ1NextRead ) )
   PRINT "EQ1WriteFlag:" + STR$(EQ1WriteFlag) + ", EQ1ReadFlag:" +
STR$(EQ1ReadFlag)

   PRINT "Queue:" + EQ1$(1) + ", QueWrite:"+ STR$(EQ1NextWrite) + ",
QueRead:" + STR$(EQ1NextRead)
   PRINT "QueRead:" + STR$( EQ1Read() )

'END Test Code

'============================================================================
===
FUNCTION EQ1Read()

   'GLOBAL EQ1$(1), EQ1Length, EQ1WriteFlag, EQ1ReadFlag, EQ1NextWrite,
EQ1NextRead

   'Has entire circular event queue previously been read ?
   IF NOT(EQ1WriteFlag XOR EQ1ReadFlag)  AND (EQ1NextWrite = EQ1NextRead)
THEN

   'Yes, return NULL (undefined) event.
      EQ1Read = 0

   ELSE
      'No, return next consecutive unread byte.
      EQ1Read = ASC( MID$( EQ1$(1), (EQ1NextRead+1), 1 ) )
      'Point to next byte location to read in circular buffer
```

```
      EQIncr( EQ1NextRead, EQ1ReadFlag, EQ1Length )

   ENDIF

END FUNCTION   'EQ1Read

'=============================================================================
===
FUNCTION EQ1Write( EQEvent )

   'GLOBAL EQ1$(1), EQ1Length, EQ1WriteFlag, EQ1ReadFlag, EQ1NextWrite,
EQ1NextRead

   'Has entire circular event queue previously been written to ?

   IF (EQ1WriteFlag XOR EQ1ReadFlag)  AND (EQ1NextWrite = EQ1NextRead)  THEN
      'Yes, failed to write byte, event queue full.
      EQ1Write = 0

   ELSE
      'No, write to next free location in circular buffer.
      EQ1$(1) = LEFT$(EQ1$(1), EQ1NextWrite) + CHR$(EQEvent) + RIGHT$(
EQ1$(1), (EQ1Length - EQ1NextWrite - 1) )

      'Point to next byte location to write in circular event queue.
      EQIncr( EQ1NextWrite, EQ1WriteFlag, EQ1Length )

      EQ1Write = 1  'Successful in writing byte.
   ENDIF

END FUNCTION   'EQ1Write

'=============================================================================
===

SUB EQ1WriteSuccess( EQEvent )

   IF EQ1Write( EQEvent ) = 0  THEN
      PRINT "Unable to store event " + STR$(EQEvent) + ", event queue full"
   ENDIF
END SUB   'EQ1WriteSuccess

'=============================================================================
===

SUB EQIncr( NextLoc, Flag, Length )

   ' Point to next circular event queue location.
   NextLoc = (NextLoc + 1)

   'Has next circular event queue location exceeded physical queue length ?
```

```
   IF NextLoc = Length THEN

     'Yes, point to first physical queue location.
      NextLoc = 0
      'Toggle flag bit to indicate location pointer
      ' is back to the beginning of the queue.
      Flag = (Flag XOR 1)
   ENDIF

END SUB   'EQIncr

'====================================================================================
FUNCTION EQEvntRdy( EQWriteFlag, EQ1ReadFlag, EQNextWrite, EQNextRead )

   'GLOBAL EQ1WriteFlag, EQ1ReadFlag, EQ1NextWrite, EQ1NextRead

   EQEvntRdy = (EQWriteFlag XOR EQ1ReadFlag) OR (EQNextWrite <> EQNextRead)
END FUNCTION   'EQEvntRdy()

'====================================================================================
```

EVENTQ2.lib:

```
'File....: EventQ2.Lib
'
'Author..: Simon Whittam
'
'Email...: s.whittam(at)xnet.co.nz
'
'Date....: 8th September 2013
'
'Language: Maximite BASIC, v4.4
'
'Purpose.: Create and initialise an instance of a First In, First Out
circular event queue.
'          The event queue saves single bytes in a small String Array and
does not
'          overwrite unread bytes.
'
'          Intended for use with a event driven Finite State Machine.
'
'Version.: v1.1
'
'License.: Attribution-NonCommercial-ShareAlike 3.0 Australia (CC BY-NC-SA
3.0 AU)
'
```

```
'Ref.....: https://en.wikipedia.org/wiki/Circular_buffer
'          http://geoffg.net/MonoMaximite.html
'          http://mmbasic.com/
'          http://creativecommons.org/licenses/by-nc-sa/3.0/au/
'
'================================================================================

'BEGIN EventQ2.Lib - Initialisation

   'Copy the "EventQ2.Lib - Initialisation" section to the beginning of the
Main BASIC module.
   'Unremark the NEXT line when using this file as as library file
   'LIBRARY LOAD "EventQ2.Lib"

   OPTION BASE 0     'event queue data referenced as an offset from: 0 -
(Length -1)

   'Configure event queue as empty
   EQ2Length    = 4  'Event queue length.
   EQ2WriteFlag = 1  'Gets toggled after writing to last location in event
queue.
   EQ2ReadFlag  = 1  'Gets toggled after reading last location in event
queue.
   EQ2NextWrite = 0  'Next location to write a byte.
   EQ2NextRead  = 0  'Next location to read a byte.

   DIM EQ2$(1) LENGTH EQ2Length  'Create a circular event queue using a
string/byte array.

   EQ2$ = "FIFO"

'END EventQ2.Lib - Initialisation

'================================================================================

'BEGIN Test code

   'Note:
   '   For an empty queue, the following variables are initialised as
follows:
   '         EQ2WriteFlag = 1
   '         EQ2ReadFlag  = 1
   '         EQ2NextWrite = 0
   '         EQ2NextRead  = 0
   '


   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQWrite:" + STR$( EQ2Write( 49 ) )
```

```
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQWrite:" + STR$( EQ2Write( 50 ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQWrite:" + STR$( EQ2Write( 51 ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQWrite:" + STR$( EQ2Write( 52 ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
'   PRINT "EQWrite:" + STR$( EQ2Write( 53 ) )
   EQ2WriteSuccess(53)
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQ2WriteFlag:" + STR$(EQ2WriteFlag) + ", EQ2ReadFlag:" +
STR$(EQ2ReadFlag)

   PRINT

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)
   PRINT "QueRead:" + STR$( EQ2Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)
   PRINT "QueRead:" + STR$( EQ2Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)
```

```
   PRINT "QueRead:" + STR$( EQ2Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)
   PRINT "QueRead:" + STR$( EQ2Read() )

   PRINT  "EvntRdy:" + STR$( EQEvntRdy( EQ2WriteFlag, EQ2ReadFlag,
EQ2NextWrite, EQ2NextRead ) )
   PRINT "EQ2WriteFlag:" + STR$(EQ2WriteFlag) + ", EQ2ReadFlag:" +
STR$(EQ2ReadFlag)

   PRINT "Queue:" + EQ2$(1) + ", QueWrite:"+ STR$(EQ2NextWrite) + ",
QueRead:" + STR$(EQ2NextRead)
   PRINT "QueRead:" + STR$( EQ2Read() )

'END Test Code

'=============================================================================
===
FUNCTION EQ2Read()

   'GLOBAL EQ2$(1), EQ2Length, EQ2WriteFlag, EQ2ReadFlag, EQ2NextWrite,
EQ2NextRead

   'Has entire circular event queue previously been read ?
   IF NOT(EQ2WriteFlag XOR EQ2ReadFlag)  AND (EQ2NextWrite = EQ2NextRead)
THEN

   'Yes, return NULL (undefined) event.
      EQ2Read = 0

   ELSE
      'No, return next consecutive unread byte.
      EQ2Read = ASC( MID$( EQ2$(1), (EQ2NextRead+1), 1 ) )
      'Point to next byte location to read in circular buffer
      EQIncr( EQ2NextRead, EQ2ReadFlag, EQ2Length )

   ENDIF

END FUNCTION   'EQ2Read

'=============================================================================
===
FUNCTION EQ2Write( EQEvent )

   'GLOBAL EQ2$(1), EQ2Length, EQ2WriteFlag, EQ2ReadFlag, EQ2NextWrite,
EQ2NextRead

   'Has entire circular event queue previously been written to ?
```

```
   IF (EQ2WriteFlag XOR EQ2ReadFlag)  AND (EQ2NextWrite = EQ2NextRead)  THEN
      'Yes, failed to write byte, event queue full.
      EQ2Write = 0

   ELSE
      'No, write to next free location in circular buffer.
      EQ2$(1) = LEFT$(EQ2$(1), EQ2NextWrite) + CHR$(EQEvent) + RIGHT$(
EQ2$(1), (EQ2Length - EQ2NextWrite - 1) )

      'Point to next byte location to write in circular event queue.
      EQIncr( EQ2NextWrite, EQ2WriteFlag, EQ2Length )

      EQ2Write = 1  'Successful in writing byte.
   ENDIF

END FUNCTION   'EQ2Write

'=============================================================================
===

SUB EQ2WriteSuccess( EQEvent )

   IF EQ2Write( EQEvent ) = 0  THEN
      PRINT "Unable to store event " + STR$(EQEvent) + ", event queue full"
   ENDIF
END SUB   'EQ2WriteSuccess

'=============================================================================
===
'
'SUB EQIncr( NextLoc, Flag, Length )
'
'   ' Point to next circular event queue location.
'   NextLoc = (NextLoc + 1)
'
'
'   'Has next circular event queue location exceeded physical queue length ?
'
'   IF NextLoc = Length THEN
'
'     'Yes, point to first physical queue location.
'      NextLoc = 0
'
'      'Toggle flag bit to indicate location pointer
'      ' is back to the beginning of the queue.
'      Flag = (Flag XOR 1)
'   ENDIF
'
'END SUB   'EQIncr
'
'=============================================================================
```

```
===
'
'FUNCTION EQEvntRdy( EQWriteFlag, EQReadFlag, EQNextWrite, EQNextRead )
'
'   EQEvntRdy = (EQWriteFlag XOR EQReadFlag) OR (EQNextWrite <> EQNextRead)
'
'END FUNCTION   'EQEvntRdy()
'
'=============================================================================
===
```

From:
https://fruitoftheshed.com/wiki/ - **FotS**

Permanent link:
https://fruitoftheshed.com/wiki/doku.php?id=mmbasic_original:mm_event

Last update: **2024/01/19 09:39**